Title

Student student name
Student ID ID

1 作业说明

1.1 作业总体说明:

- 提交内容: 中文、不少于 5 页、latex 编译成 pdf。
- 实现并撰写一篇报告: 包含引言、相关工作介绍、提供清晰的数学推导、伪代码。实现 multi-head self-attention、position-wise FFN、残差 +LayerNorm、位置编码等。完成后分数 60-70 分
- 手工搭建一个 Transformer，在小规模文本建模任务上训练并做消融实验, 包含框架说明、关键实现片段、实验设置、结果图表）（如果只有 encoder block 70-80 分，加 decoder block 80-90 分）
- 代码需开源并附上运行说明 (10 分）

1.2 代码开源要求

- 一个 Git 仓库（GitHub link）包含: src/、requirements.txt、README.md、scripts/run.sh。
- results/ 放训练曲线图与表格。
- 在 README 写清楚运行命令与硬件要求，给出重现实验的 exact 命令行（含随机种子）。

1.3 小数据集验证

- 基础: 能在小数据集上跑通训练（loss 下降）; 代码（含 README）。
- 进阶: 实现训练稳定性技巧（学习率调度、梯度裁剪、AdamW）、参数统计、模型保存/加载、训练曲线可视化。
- 挑战: 实现 decoder、实现相对位置编码、实现稀疏/线性注意力或并行化优化、做规模/超参敏感性分析、在更大数据上微调或蒸馏。

1.4 数据集合建议

- 验证数据集合可以自由选择，建议数据集合小一些。在报告中将数据集介绍写明并附链接，在代码仓库中提供数据集压缩包。
- Language Modeling Tasks (Encoder-only Transformer): Exemplifying some datasets suitable for training small-scale encoder-only Transformers for language modeling tasks. Each dataset is lightweight and accessible via the Hugging Face Datasets platform.
- Sequence-to-Sequence Tasks (Encoder–Decoder Transformer): Exemplifying some datasets are commonly used for sequence-to-sequence learning tasks such as machine translation and summarization. They provide paired input–output text samples suitable for training encoder–decoder Transformer architectures.

Dataset	Task Type	Typical Size	Hugging Face Link	Notes
WikiText-2	Word-level LM	∼2M tokens	wikitext	Classic small dataset; easy to tokenize and train small models.
Penn Treebank (PTB)	Word-level LM	∼1M tokens	ptb_text_only	Extremely compact; good for debugging models.
Tiny Shakespeare	Character-level LM	∼1 MB	tiny_shakespeare	Ideal for first training test; converges quickly.
AG News Subset	Text classification	∼120K samples	ag_news	Can be treated as classification if you prefer supervised fine-tuning.

表 1: Language modeling and classification datasets for encoder-only Transformer training.

Dataset	Task Type	Typical Size	Hugging Face Link	Notes
IWSLT2017 (EN↔DE)	Machine Translation	∼200K pairs	iwslt2017	Excellent for small-scale MT; official benchmark.
TED Talks (Multi-lingual)	Translation / Paraphrase	∼100K pairs	ted_talks_iwslt	Small and language-diverse.
Gigaword Subset	Summarization	∼200K pairs	gigaword	Works well for headline generation.
CNN/DailyMail (trimmed)	Summarization	∼300K pairs	cnn_dailymail	Use subset of 10K–20K for local training.

表 2: Sequence-to-sequence datasets for encoder–decoder Transformer training.

1 Introduction

Introduce the background and motivation of the Transformer architecture. Explain the purpose of this assignment and summarize the main contributions of your implementation.

- What problem does the Transformer solve?
- Why is it important to build it from scratch?
- What are your goals (e.g., understanding self-attention, reproducing training behavior)?

2 Related Work

Briefly review the Transformer model and related architectures (e.g., different attention mechanisms). Cite the original Transformer paper [1] and optionally more recent improvements.

3 Model Architecture and Mathematical Derivation

Provide the theoretical foundation and equations behind each module. All subsections are examples.

3.1 Scaled Dot-Product Attention

Present the attention formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Explain all symbols and shapes.

3.2 Multi-Head Attention

Derive how multiple attention heads are combined. Explain why multi-head improves representation capacity.

3.3 Position-Wise Feed-Forward Network

Define the two-layer MLP applied independently to each token.

3.4 Residual Connections and Layer Normalization

Discuss the role of residual connections and normalization in stabilizing training.

3.5 Positional Encoding

Describe sinusoidal or learned positional encodings. Provide the mathematical formula.

4 Implementation Details

Describe how you implemented the model and training pipeline.

- Framework and language (e.g., PyTorch)
- Implementation of attention, FFN, normalization
- Masking (padding mask, future mask if decoder)
- Model hyperparameters
- Pseudocode or key code snippets

Listing 1: Simplified self-attention implementation in PyTorch

```
def scaled_dot_product_attention(Q, K, V, mask=None):
    d_k = Q.size(-1)
    scores = Q @ K.transpose(-2, -1) / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)
    attn = torch.softmax(scores, dim=-1)
    return attn @ V, attn
```

5 Experimental Setup

Describe the dataset, training setup, and evaluation metrics.

- Dataset (e.g., WikiText-2, small text corpus)
- Data preprocessing
- Training hyperparameters: batch size, learning rate, optimizer, scheduler, number of epochs
- Evaluation metrics (loss, perplexity, accuracy)

Provide a clear hyperparameter table:

表 3: Hyperparameter Settings

Parameter	Value
Embedding dimension	128
Number of heads	4
Feed-forward dimension	512
Number of layers	2
Batch size	32
Learning rate	3e-4
Optimizer	Adam

6 Results and Analysis

Present quantitative and qualitative results.

- Training and validation curves
- Sample predictions or generated text
- Comparison between variants (e.g., different number of heads)
- Ablation study: what happens if positional encoding is removed?

Discuss findings and insights.

7 Reproducibility and Code Structure

Explain how to reproduce the results.

- github link
- Dependencies and environment setup
- Folder structure of your code repository
- Command line examples
- Expected runtime and hardware used

Example:

Listing 2: Example training command

```
$ conda create -n transformer python=3.10
$ pip install torch matplotlib
$ python train.py --config configs/base.yaml
```

8 Conclusion and Future Work

Summarize what you have implemented and learned. Discuss possible extensions:

- Adding a decoder for sequence-to-sequence tasks
- Trying relative positional encoding
- Comparing with official PyTorch implementation

参考文献

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.